A Brief Review on the Development of HYPRE for GPUs

FASTMath4 Institute - All-Hands Meeting – Argonne National Laboratory

June 11

Ruipeng Li

LLNL-PRES-777450

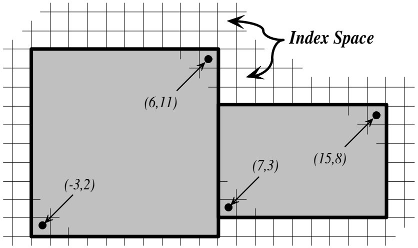
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

GPU support in HYPRE

- GPU support has been available in recent releases of HYPRE
  - enabled by various approaches including CUDA, OpenMP 4.5, RAJA, Kokkos
- Structured multigrid solvers: SMG, PFMG
  - Structured data types: grid, box, stencil, struct matrix and vector...
  - Computations are performed in BoxLoops
  - Completely ported to GPUs, MG setup and solve, in GPU device memory
- Unstructured algebraic multigrid: BoomerAMG
  - Unstructured data types: ParCSR matrices, Parvectors, ...
  - Solve phase has been ported: MATVEC, vector operations, and appropriate relaxations run on GPUs with unified memory
  - Setup phase: performed on CPUs with unified memory
  - Current focus: AMG setup phase on GPUs

Structured interface of HYPRE

- Provides access to the structured solvers: SMG, PFMG, ...
- Struct grid is composed of **boxes**



- Struct matrix and vector sit on top of struct grid
- Struct computations are performed via **BoxLoops**

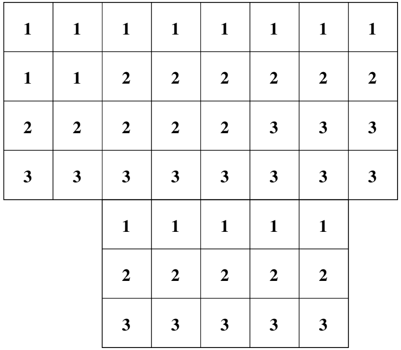
HYPRE loop abstraction for computations: BoxLoops

- Example: $\boldsymbol{y} := \boldsymbol{\alpha y}$, $\boldsymbol{y}$ is a struct vector

```
hypre_ForBoxI(i, boxes) {
    ...
    hypre_BoxLoop1Begin(dim, loop_size, data_box, start, unit_stride,
      yi);
    [#pragma omp parallel for private(yi)]
    hypre_BoxLoop1For(yi) {
        yp[yi] *= alpha;
    }
    hypre_BoxLoop1End(yi);
}
```

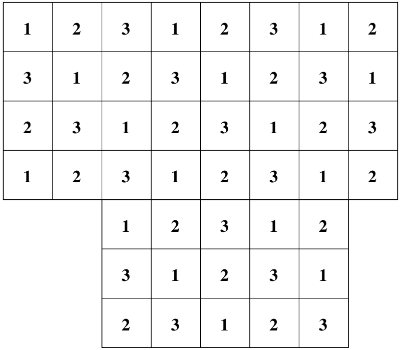
Different ideal memory access patterns

CPU: coarse-grained parallelism



CPU parallel BoxLoop

GPU: fine-grained parallelism



GPU parallel BoxLoop

GPU BoxLoops have the same interface as the CPU ones

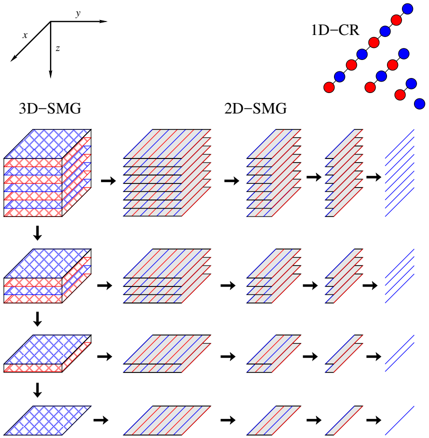
- Offload the OMP parallel region to GPU: with CUDA or OpenMP 4.5

```
#define hypre_BoxLoop1Begin(ndim,loop_size,dbox1,start1,stride1,i1) {\
  /* host code: */ \
  hypre_BoxLoopDeclareInit(ndim,loop_size) \
  hypre_BoxKDeclareInit(1,start1,dbox1,stride1) \
  /* device code (CUDA): */ \
  BoxLoopforall(hypre_exec_policy,tot,HYPRE_LAMBDA(HYPRE_Int idx) {
    hypre_BoxLoopSet1(i1)
```

```
  /* device code (OMP4.5): */ \
  _Pragma(omp target teams distribute parallel for) \
  for (thread=0; thread<tot; thread++) {\
    hypre_BoxLoopSet1(i1)
```

- also available with RAJA or Kokkos

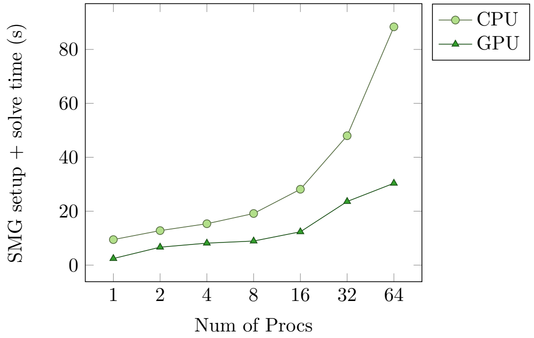
Complicated structured MG solvers can be ported seamlessly

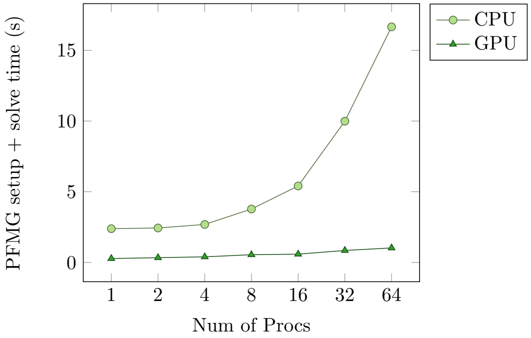


- 3-D SMG semicoarsens in the $\boldsymbol{z}$-direction and uses $\boldsymbol{xy}$-plane smoothing by one V-cycle of 2-D SMG, which semicoarsens in the $\boldsymbol{y}$-direction and uses 1-D line smoothing in the $\boldsymbol{x}$-direction by Cyclic Reduction
- Interpolation operator is computed by performing a sequence of (simultaneous) SMG solves (of one dimension lower)
- SMG is recursively constructed and is applied in the solve phase
- SMG(3D) $\longrightarrow$ SMG(2D) $\longrightarrow$ CR(1D)

Performance of SMG/PFMG on GPUs

- 3-D Poisson problem. Problem size $\mathbf{200^3} \times \boldsymbol{N_p}$
- Running on **ray** at LLNL. IBM Power8 + 4 NVIDIA Tesla P100 (Pascal) per node. CUDA 9.2, IBM XL compilers



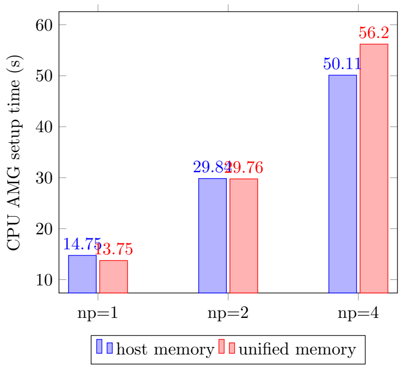


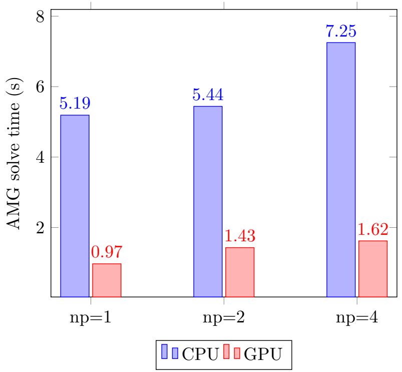
The solve phase of BoomerAMG has been ported to GPUs

- GPU kernels for MATVEC and vector operations have been implemented with CUDA or OpenMP 4.5
  - with the option of using cuSPARSE and cuBLAS
- Several GPU-appropriate smoothers have been implemented
  - L_1 Jacobi and polynomial smoothers
- Setup phase is much more complicated. Currently, remains on CPUs with CUDA unified memory

Performance of AMG on GPUs

- 3-D Poisson problem. Problem size $\mathbf{200^3} \times N_p$
- Running on **ray** at LLNL. IBM Power8 + 4 NVIDIA Tesla P100 (Pascal) per node. CUDA 9.2, IBM XL compilers



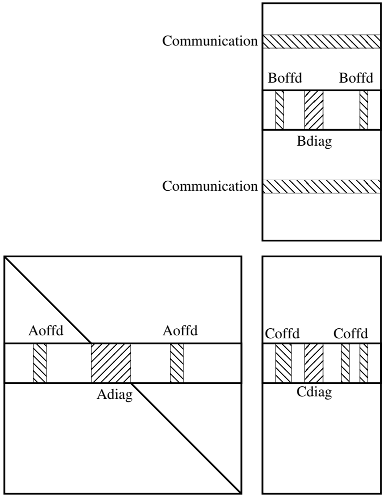


Work in progress: AMG setup on GPUs

- Main ingredients of classical AMG setup
  - Compute SoC matrix $\boldsymbol{S}$. Relatively easy to implement
  - Compute coarsening $\boldsymbol{C}/\boldsymbol{F}$. PMIS algorithm has been implemented
  - Compute interpolation $\boldsymbol{P}$. Direct interpolation has been implemented
  - Compute Galerkin product $\boldsymbol{RAP}$. Most difficult. Two algorithms have been implemented. Computed in pairs: $\boldsymbol{Q} = \boldsymbol{AP}$ and $\boldsymbol{RAP} = \boldsymbol{RQ}$

Distributed SpGEMM in ParCSR

- In parallel CSR, each process owns a slice of rows partitioned into diag and offd parts
- Decompose ParCSR $\boldsymbol{A}$ × ParCSR $\boldsymbol{B}$ into *local* CSR matrix operations: multiplication, (partial) addition, splitting, merging, and transposition for $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{B}$
- Also wrote CUDA kernels other than multiplication: with **Thrust**; much simpler
- (GPU) Communications are involved for sending/receiving *external* rows, and can be overlapped with computations



A complete hash-table based SpGEMM algorithm

➤ Compute SpGEMM in **4** steps

❶ Row NNZ estimation: to allocate “reasonable-sized” hash tables for ❷;

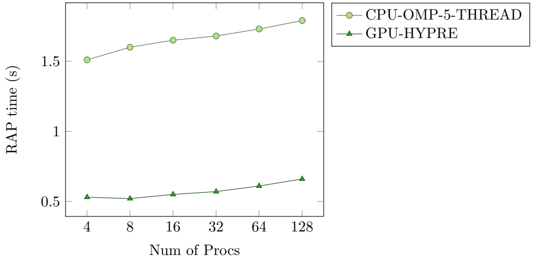
❷ Symbolic analysis: to compute row counts (bounds) and row pointers $\boldsymbol{ic}$, and to allocate “adequate-sized” hash tables for ❸;

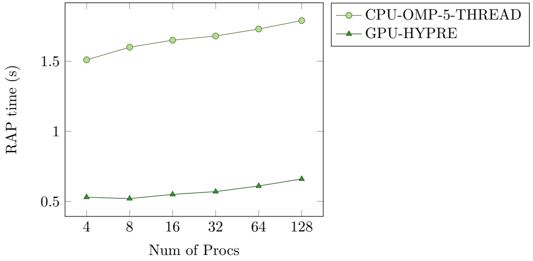
❸ Numeric multiplication: to compute the column indices and values in $\boldsymbol{jc}$ and $\boldsymbol{c}$;

❹ Post-processing: to remove the gaps between rows computed from ❸.

GPU ParCSR SpGEMM: 3-D 27-pt Laplacians

- BoomerAMG with HMIS coarsening and ext+i interpolation
- Compute $\boldsymbol{P}^{\mathsf{T}}\boldsymbol{AP}$ on the 1st level of AMG
- Weak scalability study. Local problem size: $\mathbf{128^3}$
- ray: IBM Power8 + 4 NVIDIA P100. lassen: IBM Power9 + 4 NVIDIA V100
- On ray 4 – 64 GPUs (left) and lassen 4 – 128 GPUs (right). y-axis: time





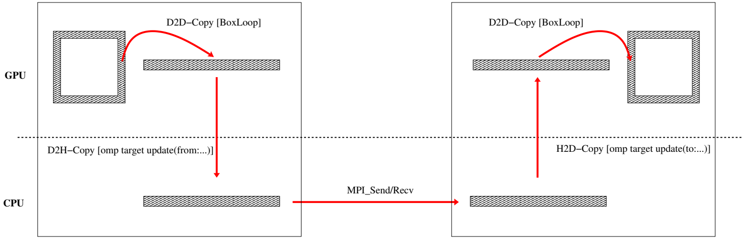
HYPRE's memory model

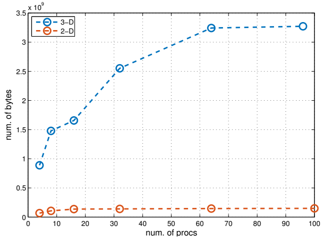
- Three conceptual memory locations
  - `HYPRE_MEMORY_HOST`, `HYPRE_MEMORY_DEVICE`, `HYPRE_MEMORY_SHARED`
- Mapped to different physical memory in different configurations

	`HYPRE_MEMORY_HOST`	`HYPRE_MEMORY_DEVICE`	`HYPRE_MEMORY_SHARED`
c1	HOST	HOST	HOST
c2	HOST	CUDA DEVICE	CUDA DEVICE
c3	HOST	CUDA DEVICE	CUDA MANAGED

  - c1: non-GPU configuration
  - c2: `--with-cuda`, `--with-device-openmp`
  - c3: c2 + `--enable-unified-memory`
- `hypre_TAlloc(HYPRE_Complex, n, HYPRE_MEMORY_DEVICE);`

GPU-GPU communications through MPI


GPU
CPU
D2D–Copy [BoxLoop]
D2D–Copy [BoxLoop]
D2H–Copy [omp target update(from:...)]
H2D–Copy [omp target update(to:...)]
MPI_Send/Recv



- Memory transfer: GPU $\rightarrow$ GPU $\rightarrow$ CPU $\xrightarrow{\text{MPI}}$ CPU $\rightarrow$ GPU $\rightarrow$ GPU
- GPU aware MPI can help reduce GPU - CPU communication cost
- Communication volume in 3-D is much higher than that in 2-D

Conclusion

- The structured and unstructured algebraic multigrid solvers of HYPRE have been enabled on GPUs by different approaches
- Structured MG solvers have both the setup and solve phases on GPUs, whereas the setup phase of AMG remains on CPUs
- AMG setup on GPUs is work in progress. Major components have been implemented
- HYPRE's abstract memory model enables execution on heterogeneous platforms

CASC
Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

THANK YOU!

Questions & Comments

`li50@llnl.gov`